MEAL DELIVERY OPTIMIZATION MODEL

San Francisco State University

Submitted by:

Padmasree Sappa

DS 852 Managerial Decision Making

05.15.2025

Table of Contents

Meal Delivery Optimization Model (MDOM)

Optimization of Courier Assignments in Meal Delivery Services

1 Introduction – Dataset link

Meal delivery services operate in a high-pressure environment where time is the most critical factor. Customers expect fast and accurate deliveries, and even small delays can lead to dissatisfaction, complaints, and loss of brand trust. Operationally, delivery companies also face the challenge of balancing courier workloads, minimizing travel times, and managing external factors like traffic and weather while controlling costs.

To tackle these complex issues, we developed the Meal Delivery Optimization Model (MDOM). This model uses Excel Solver, a powerful optimization tool, to assign delivery personnel to customer orders. Instead of relying on random or manual courier matching, MDOM systematically selects the best available courier for each order based on real-world conditions. Factors like traffic congestion, courier speed, distance, and weather impact are mathematically incorporated into the model, ensuring that every assignment is both logical and efficient.

The core idea behind MDOM is to **deliver food to the customer as fast as possible, using the optimal courier under real-world operating conditions.** When deliveries are fast and more predictable, customer satisfaction improves, operational efficiency increases, and the overall service quality strengthens.

2 Problem Statement

The main problem faced by meal delivery services is the **inefficient assignment of couriers to customer orders**. Without an optimized system, courier assignments can become random and unbalanced. This leads to:

- Longer delivery times
- Higher operational costs
- Unfair courier workloads
- Dissatisfied customers

In real-world terms, imagine two couriers available at the same time. **Courier A** rides a motorcycle and averages **40 km/h**, and **Courier B** uses a scooter and averages **35 km/h**. In heavy traffic, even a small speed difference can impact delivery time by several minutes. Assigning the wrong courier to a faraway customer would result in a significant delivery time increase.

Our model solves this problem by **automatically matching the right courier to each order**, based on factors like speed, distance, traffic, and weather conditions, thus minimizing the total delivery time for all orders within a batch. **In short, pick the best courier for each order to save time and keep customers happy.**

3 Project Objectives

This project was designed with a focused set of objectives to make the model practical and impactful:

- **Minimize Total Delivery Time:** The primary objective is to reduce the overall delivery time by matching couriers to orders. Solver is set up to find the combination of assignments that gives the shortest total delivery time.
- **Assign Each Order Exactly Once:** Each customer order must be assigned to only one courier. No order can be missed, left out, or assigned to multiple couriers.
- **Balance Courier Workloads:** The model must assign orders fairly and efficiently. Couriers should not be overloaded with too many orders. Each courier is assigned to at least one order.
- **Account for Real-World Constraints:** The model incorporates important factors that significantly impact delivery time, such as traffic conditions, weather disruptions, and courier speed.

These objectives ensure that the model is practical in real delivery operations.

4 Dataset Description

The success of any optimization model depends heavily on the quality of the data it is built. For this project, we worked with real-world operational data collected from meal delivery services. This data provides all the necessary information for the model and helps validate the model's applicability. The **dataset** included the following critical information:

- **Customer Orders**:
  - **Unique Order IDs** to identify each customer request.
  - **Customer locations** by latitude and longitude coordinates
- **Restaurant Details**:
  - **Location** by latitude and longitude coordinates
- **Delivery Person Details**:
  - **Courier IDs**

- **External Conditions**:
    - **Traffic Conditions** at the time of order (e.g., Low, Medium, Jam)
    - **Weather Conditions** (e.g., Sunny, Rainy, Foggy)
- **Timing Details**:
    - **Order placement time**, **pickup time**, and **delivery time** — used to understand service windows and compute realistic delivery targets.

Each record represented one unique delivery attempt. With this set of attributes, we could calculate distances, estimate travel times, and factor external conditions, leading to optimal courier assignments.

5 Data Cleaning

Why was data cleaning important?

Optimization models like Solver require complete, accurate, and logical datasets to work correctly. If there are missing values, unrealistic entries, or inconsistent formats, Solver will either crash, produce wrong results, or suggest impractical courier assignments. Therefore, cleaning the data was a critical first step to ensure that the model's outputs would be reliable.

What cleaning steps were implemented?

We carefully processed the raw dataset through the following steps:

1) **Removed Missing Values:** Any row with missing critical fields, such as Delivery Person ID, Customer Coordinates, Order Times, or Ratings, was dropped.

2) **Deleted Duplicates:** Duplicate records were removed to avoid biasing the model with repeated deliveries.

3) **Age Validation:** Only couriers aged between 18 and 65 years were retained, representing the realistic working age for delivery operations.

4) **Rating Validation:** Only couriers with ratings between 3.9 and 5.0 were kept, ensuring that only reliable, high-performing couriers were assigned orders.

5) **Outlier Removal:** Extremely long deliveries (> 200 minutes) or excessive delivery distances (> 50 kilometers) were treated as outliers and removed.

6) **Standardized Text Fields:** Extra spaces in text fields (like city names and traffic conditions) were cleaned up to ensure smooth dropdowns, VLOOKUPs, and formulas later.

After cleaning, we had a pure, consistent, and ready-to-use dataset, ensuring the optimization model would perform correctly and reflect a real-world condition.

6 Data Manipulation and Feature Engineering

Once we had a clean and reliable dataset, the next step was preparing the data to make it useful for optimization. Raw data alone was not enough — we needed to engineer new fields as model inputs

6.1 Distance Calculation between the Restaurant and the Customer

To accurately model delivery time, we first needed to know how far the customer was from the restaurant. Simply using a straight-line or "city block" distance would have been inaccurate since such approaches do not factor in the Earth's curvature, resulting in an underestimated distance, which would lead to an inaccurately calculated delivery time. We instead applied the Haversine Formula, which calculates the great-circle distance between two points on the Earth's surface based on latitude and longitude. The Excel version of the Haversine Formula is shown as follows:

$$\begin{gathered} Distance\ (km) = 6371 \times ACOS(cos\ (RADIANS(lat1)) \times cos\ (RADIANS(lat2)) \\ \times cos\ (RADIANS(lon2) - RADIANS(lon1)) \\ + sin\ (RADIANS(lat1)) \times sin\ (RADIANS(lat2))) \end{gathered}$$

Example:

- Restaurant at (28.61, 77.23)
- Customer at (28.64, 77.22)

$$\begin{gathered} Distance\ (km) = 6371 \times ACOS(cos\ (RADIANS(28.61)) \times cos\ (RADIANS(28.64)) \\ \times cos\ (RADIANS(77.22) - RADIANS(77.23)) \\ + sin\ (RADIANS(28.61)) \times sin\ (RADIANS(28.64))) \end{gathered}$$

As a result, the distance from the Haversine Formula is approximately 3.5 km

6.2 Estimated Delivery Time Calculation

Distance alone doesn't determine delivery time — traffic, weather, and courier speed also affect how long it takes to deliver a particular order. Thus, we incorporated these factors in the delivery time estimation.

$$Estimated\ Time\ (mins) = Prep.Time + \frac{60 \times Distance}{Traffic\ Factor \times Weather\ Factor\ \times Vehicle\ Speed}$$

Where:

- **Prep Time**: Time needed to prepare the meal at the restaurant.
- **Distance**: Travel distance from restaurant to customer (calculated earlier).
- **Vehicle Speed**: Average speed of each courier based on their vehicle type.

- **Traffic Factor**: Adjusts speed down during congestion (low = 1, medium = 1.3, high = 1.6 and jam = 2).

- **Weather Factor**: Adjusts speed based on weather conditions (sunny = 1, windy = 1.1, cloudy = 1.2, stormy = 1.3, fog = 1.4 and sandstorms = 1.5)

Example:
- Distance = 3 km
- Average Courier Speed = 5 km/h
- Traffic Factor = 1.3 (Medium Traffic)
- Weather Factor = 1.2 (Cloudy)
- Prep Time = 10 minutes

Calculation:

$$Estimated\ Travel\ Time = 10 + \frac{(60 \times 3)}{1.3 \times 1.2 \times 5} = 33.1\ minutes$$

Thus, the expected delivery would take about 33 minutes from order placement to the customer's doorstep.

6.3 Encoding Categorical Variables

Traffic conditions ("Low", "Medium", “High”, and "Jam") and weather conditions (“Sunny”, “Windy”, “Cloudy”, “Stormy”, “Fog”, and “Sandstorms”) were text fields. Solver cannot process text, so we numerically encoded it:

- Traffic Density Encoding:
    - Low → 1.0
    - Medium → 1.3
    - High → 1.6
    - Jam → 2.0

- Weather Conditions Encoding:
    - Sunny → 1.0
    - Rainy → 1.2
    - Foggy → 1.4

By converting these into numeric multipliers, the model can include these factors in the calculation.

6.4 Extraction of Unique IDs

To build dynamic assignment matrices, we needed clean lists of unique:

- **Order IDs**
- **Delivery Person IDs**

We used the Excel =UNIQUE() formula to remove duplicate IDs. This allowed the model to adapt automatically whenever a new batch (city, date, slot) was selected without manually editing anything.

6.5 Helper Columns

Finally, to support calculations, we created helper columns that dynamically adjusted fields like:

- Adjusted Speed (based on traffic and weather)
- Adjusted Distance
- Effective Travel Time

These helper columns made sure that the Estimated Delivery Time formula stayed clean and modular instead of bloating into one messy formula.

7 Mathematical Model

Sets and Indices:

- $i \in Orders : Index\ for\ customer\ orders$
- $j \in Couriers : Index\ for\ available\ delivery\ personnel$

Parameters:

- $tij = Estimated\ delivery\ time\ if\ courier\ j\ delivers\ order\ i$ (calculated based on distance, speed, traffic, and weather)
- $p = Penalty\ time\ per\ extra\ assignment\ (optional)$

Decision Variables:

- $x_{ij} = \begin{cases} 1\ if\ courier\ j\ is\ assigned\ to\ order\ i\ ; \\ 0\ otherwise \end{cases}$

Objective Function:
Minimize the total estimated delivery time across all orders and couriers.

$$Minimize \sum_i \sum_j t_{ij} * x_{ij} + K(M - N)$$

where

1. t_{ij} Represents the estimated delivery time considering distance, traffic, weather, and average courier speed.

2. $K(M - N)$ Represents penalty time, which is incurred during a time slot if there's a variation in the number of orders delivered by different couriers.

$$\boxed{\begin{aligned} penalty\ time &= penalty\ time\ factor \times (max\ deliveries\ per\ courier \\ &\quad - min\ deliveries\ per\ courier) \\ penalty\ time &= K(M - N) \end{aligned}}$$

K represents the penalty time factor (default: 25 mins)

M represents the maximum number of deliveries per courier

N represents the minimum number of deliveries per courier

Constraints:

1. **Order Assignment Constraint:**

Each customer order must be assigned to exactly one courier

$$\sum_j x_{ij} = 1 \quad for\ all\ i$$

2. **Courier Load Constraint:**

Each courier is assigned at least one order

$$\sum_i x_{ij} \geq 1 \quad for\ all\ j$$

3. **Binary Assignment Constraint**

$$x_{ij} \in \{0,1\}\ \ for\ all\ i\ and\ j$$

8 VBA Macros and Automation

Setting up the optimization model manually for every new batch of orders and couriers would have been extremely tedious and error-prone. To make the system fast, reliable, and user-friendly, we automated critical tasks using VBA Macros in Excel.

8.1 Building the Delivery Assignment Matrix

This macro automatically builds the **Decision Variables Matrix**:

- It initially inserts random 0 and 1, representing unassigned (0) or assigned (1) relationships between orders and couriers.
- It automatically sets row constraints by inserting =SUM() formulas at the end of each order row. This ensures each order is assigned to exactly one courier.
- It automatically sets column constraints by inserting =SUM() at the bottom of each courier column. This tracks how many orders each courier is handling.
- It formats the matrix neatly with a light background color for easy visual tracking.

In short, with one click, your entire assignment matrix + constraint logic is ready for Solver.

8.2 Clearing Matrix Constraints

Before running a new optimization batch, old matrices and constraints must be cleared. This macro:

- Deletes all 0s, 1s, and SUM formulas from previous runs.
- Resets background colour to plain white.
- Ensures no leftover formulas interfere with the next Solver execution.

8.3 Filling the Delivery Time Matrix

The Delivery Time Matrix stores the estimated delivery times for every possible combination of orders and couriers.

This macro:

- Uses dynamic VLOOKUP formulas to fetch distance and average courier speed.
- Calculates estimated delivery time for each pair (order, courier).
- Populates the Delivery Time Matrix.

Example inserted formula:

$$= (VLOOKUP(OrderID, CleanRawData!A:U, 21, FALSE) * 1000 / VLOOKUP(DeliveryPersonID, Input!O2:R1171, 4, FALSE)) / 60$$

This formula calculates delivery time by adjusting raw distances with courier speeds.

8.4 Clearing the Delivery Time Matrix

Similar to clearing the assignment matrix, this macro:

- Clears all time estimates and formulas from the Delivery Time Matrix.

- Prepares the sheet for fresh time recalculations when a new batch is selected.

In short, the entire setup — building matrices, inserting constraints, filling times — became one-click operations thanks to macros, making the model scalable, fast, and error-free.

8.5 Solver Setup with Penalty Logic

To avoid manually configuring Solver every time, this macro:

- Sets the objective function as the total delivery time (SUMPRODUCT of both matrices).
- Adds binary constraints for assignments (0 or 1).
- Enforces row and column constraints.
- Adds a penalty formula to discourage uneven courier workloads.
- Runs Solver with the Assume Linear and Assume Non-Neg options for consistency.

Result: Solver produces optimal order–courier assignments in one click.

8.6 Email Rollout Macro

After optimization, this macro automates email communication to the delivery team.

- Compiles a clean table of assigned Order IDs, Courier IDs, Restaurant Locations, and Delivery Locations.
- Pulls the selected Date and Time Slot from the interface.
- Sends the assignment summary via Outlook to predefined recipients.

Benefit: No need to manually extract or copy results — one macro generates and sends all details instantly.

9 Excel Formulas Used

Excel formulas played a critical role in building the optimization engine behind the scenes.

Here's a detailed look at key formulas:

- **Haversine Distance Calculation**:
  - Computes the realistic distance between the restaurant and the customer.
- **Estimated Delivery Time**:

    - Combines distance, average courier speed, traffic factor, and weather factor dynamically.

- **SUMPRODUCT**:
    - Calculates the total delivery time based on current courier assignments.

$$\boxed{= SUMPRODUCT(DecisionMatrixRange, DeliveryTimeMatrixRange)}$$

- **UNIQUE()**:
    - Extracts unique Order IDs and Delivery Person IDs for dynamic matrix building.
- **IFERROR()**:
    - Prevents errors in lookup formulas, ensuring clean matrix generation.
- **Data Validation**:
    - Created dropdown menus for City, Order Date, and Time Slot selection.
- **Helper Columns**:
    - Adjusted Speed, Traffic Factors, Weather Factors, and Effective Travel Times were broken into clear helper columns to keep formulas modular and readable.

Why does this matter: Because formulas dynamically adapt as orders and couriers change, no need to manually update anything.

10 Understanding the Matrices

The optimization model revolved around two important matrices:

10.1 Decision Variables Matrix

- Rows = Customer Orders
- Columns = Available Couriers
- Each cell = 0 (not assigned) or 1 (assigned)
- Row sums = Exactly 1 (each order must have one courier)

This matrix tells Solver which courier is assigned to which order.

10.2 Delivery Time Matrix

- Rows = Customer Orders

- Columns = Available Couriers
- Each cell = Estimated time (in minutes) for that courier to deliver that order.

This matrix feeds into the SUMPRODUCT formula that Solver minimizes.

10.3 How they work together:

- Solver decides which cells get a 1 (assignment).
- SUMPRODUCT calculates the total time.

Solver tries different combinations until it finds the minimum total time without violating constraints.

11 Solver Setup

Solver was the heart of the optimization model. We set it up carefully to frame the courier assignment as an Integer Linear Programming (ILP) problem.

Key components of the Solver setup:

- **Decision Variables**:
  - Each cell in the Decision Matrix is a binary variable (0 or 1).
  - 0 = Courier not assigned to that order.
  - 1 = Courier assigned to that order.
- **Objective Function**:
  - Minimize the **total delivery time** across all orders.
  - Mathematically, Solver minimizes:

$$\boxed{SUMPRODUCT(Decision\ Matrix\ *\ Delivery\ Time\ Matrix)}$$

- **Constraints**:
  - Each order must be assigned exactly once (row sum = 1).
  - Couriers must not exceed maximum order limits (column constraints if needed).
  - Decision variables must strictly stay binary (either 0 or 1).
- **Solving Method**:

- Solver’s **Simplex LP** method was used — ideal for large, binary, linear optimization models.

In short, Solver selects a combination of 1s and 0s throughout the matrix that minimizes total delivery time while adhering to real-world assignment rules.

12 Conclusion

In conclusion, this project has successfully developed and demonstrated the Meal Delivery Optimization Model (MDOM), a robust solution addressing the critical challenges of inefficient courier assignments and lengthy delivery times prevalent in meal delivery services. By leveraging real-world operational data and incorporating methodologies such as Haversine distance calculations, the model has proven to be practical in business operations.

The MDOM's ability to translate complex logistical variables into an efficient, automated assignment process offers tangible benefits. These include not only a significant reduction in total delivery times but also the promotion of fairer workload distribution among couriers, ultimately leading to enhanced customer satisfaction and loyalty.

Therefore, the MDOM is a practical, deployable tool capable of delivering operational improvements and a competitive advantage to meal delivery businesses. This project demonstrates how data-driven optimization can solve real-world logistical problems and enhance the service landscape."